Chapter 1 Introduction

Software engineers apply methods, techniques, and related tools to understand a problem and then to elaborate a software solution, gradually, through a series of phases. Each phase in this process of software development aims to achieve one or more specific purposes and, in general, the amount of detail addressed increases as the development process moves nearer to the point where software reaches the customer. A pivotal point, early in the software development process, occurs with the creation of a software architecture.

The total investment in a particular software engineering project up through the definition of a software architecture appears rather modest when viewed as a percentage of the total software development and maintenance costs. From the point where a software design exists the investment increases rapidly as software components are designed, coded, tested, integrated and then tested some more, as documentation is produced, reviewed, and edited, and as the intended users are trained to exploit the new software. After delivery to the customer, the investment continues to accumulate over the life of the new software as errors are discovered and eliminated, and as enhancements are introduced. The software architecture then provides a technical road map to guide the bulk of the investment required to engineer a software solution. Mistakes made during the design of a software architecture and not detected until later in the software

development process tend to be costly. Experience with large software projects indicates that many defects discovered after release of a software product are due to errors made during design. Additionally, a significant portion of the cost for a software product during its entire life-cycle derives from detecting and correcting design flaws. Many of these design flaws, and their related costs, result from the current state of software design practice. Software engineering researchers, therefore, explore numerous paths to improve the productivity of software designers.

One path, considered by a number of software engineering researchers, leads toward methods for providing automated assistance to help designers create software architectures. More specifically, given a flow-graph representation of a software system, researchers propose a number of approaches for automatically generating a software architecture. Most of the proposed approaches lead to the construction of sequential designs, represented by structure charts. *Instead, the research described in subsequent chapters of this dissertation proposes, investigates, and evaluates a method for generating concurrent designs for real-time software, given a flow graph model of system behavior.* The proposed method assumes that a designer expresses system behavior through data/control flow diagrams, using the notation defined for Real-Time Structured Analysis, or RTSA. The proposed method encodes knowledge from a behavioral modeling approach, known as Concurrent Object-Based Real-Time Analysis, or COBRA, and from a design method, known as COoncurrent Design Approach for Real-Time

Systems, or CODARTS. [Gomaa93] The encoded knowledge leads directly to an automated assistant for designers of concurrent software.

Chapter 2, **Approaches to Software Design**, provides an overview of some methods, both those proposed by researchers and those practiced by software engineers, to address problems associated with software design. The chapter describes a range of methods for designing sequential software, concurrent and real-time software, and object-oriented software. In general, these design methods identify a set of products that embody a software design, along with a series of steps or activities that a human designer can follow to produce the required products at an acceptable level of quality and on a repeatable basis. In the best state of software practice today, a software designer uses one of these methods, as appropriate for the type of design desired. The chapter also considers some research intended to eliminate the need for design methods by providing automatic generation of software solutions from a statement of requirements. As explained in the chapter, a number of difficulties must be overcome before these automatic programming techniques can be made practical.

Chapter 3, **Overview of Research**, evaluates some existing approaches to automating the generation of designs from flow graphs. The chapter goes on to identify the objectives that a software designer aims to meet and then discusses the methods a designer uses to achieve those objectives. This discussion leads to three research problems.

1. How can flow-graph specifications be modeled and analyzed using a computer program to achieve the same semantic interpretation that a human designer gives to such flow graphs?

2. How can concurrent designs, and the characteristics and constraints of any intended target environments, be modeled using a computer program to approximate the semantic view used by a human designer?

3. How can the design processes and decision-making heuristics normally used by a human designer be modeled using a computer program?

This dissertation proposes a knowledge-based approach to address these research problems. Chapter 3 provides a description of the proposed approach. Subsequent chapters provide detailed solutions to each of the research problems.

Chapter 4, **A Meta-Model for Specifications**, defines and describes a meta-model for making semantic interpretations from data/control flow diagrams, and for representing certain specification addenda that cannot be represented directly on a flow diagram. The dissertation also provides, in Appendix A.1, **Axioms for Semantic Concepts** and in Appendix A.2, **Rules for Classifying Semantic Concepts,** more formal definitions for the semantic concepts introduced in Chapter 4.

Chapter 5, **A Meta-Model for Concurrent Designs**, defines and describes a semantic meta-model for representing concurrent designs. The chapter also illustrates a graphical notation for rendering concepts from the design meta-model in a diagrammatic form. In addition, the chapter includes a means for describing salient characteristics, such

as operating system services, hardware configurations, and design parameters, that can represent specific target environments.

Each of the next four chapters identifies, defines, and orders the decision-making processes needed to model a particular phase in the CODARTS design method. Each chapter also specifies and explains the rules, based upon design heuristics from the CODARTS design method, that make specific, design decisions. Chapter 6, **Task Structuring**, defines rules for forming concurrent tasks from semantic concepts in the specification meta-model. Chapter 7, **Task Interface Definition**, details rules for mapping semantic concepts from the specification meta-model into appropriate interfaces among tasks in a concurrent design. Chapter 8, **Module Structuring**, presents rules for forming information hiding modules from semantic concepts in the specification meta-model. Chapter 9, **Task and Module Integration**, details and discusses rules for integrating the separate task and module views, as created during task and module structuring, respectively. Taken together, the decision-making processes and design-decision rules explained in Chapters 6 through 9 can form the basis for an automated design generator.

Chapter 10, **A Prototype COncurrent Designer's Assistant**, describes a prototype implemented to investigate and evaluate the approach proposed in Chapter 3. As implemented, the prototype, named CODA, provides automated assistance in two main areas: specification analysis and design generation.

CODA is used to analyze specifications and to generate designs for four, real-time applications, as described in Appendix B, **Automobile Cruise Control and Monitoring System Case Study**, Appendix C, **Robot Controller Case Study**, Appendix D, **Elevator Control System Case Study**, and Appendix E, **Remote Temperature Sensor Case Study**. For each of these case studies, CODA generates, after analyzing an input data/control flow diagram, one or more concurrent designs.

Chapter 11, **Evaluation**, uses the results from the case studies to evaluate the effectiveness of the approach proposed in this dissertation. The chapter contains an evaluation of the proposed approach both against the research objectives and against other extant approaches. In addition, the chapter delineates the strengths and weaknesses of the proposed approach. The chapter also includes a detailed analysis, derived from the case studies, in order to support the evaluation.

The dissertation closes in Chapter 12, **Contributions and Future Research**, with a summary of the contributions made by the research described in the preceding eleven chapters and in the five appendices. The chapter also describes potential applications of the research and identifies areas for future, related research.